Un sistema híbrido neuro-genético para la construcción de controladores difusos (*)

Cena Marcelo, Kavka Carlos

{mcena,ckavka}@unsl.edu.ar
Grupo de Interés en Sistemas de Computación
Departamento de Informática
Universidad Nacional de San Luis
San Luis, Argentina

Fu Zhong Quian, Wu Geng Feng

University of Science and Technology
230026 Hefei
China

Resumen:

En este trabajo se presenta un modelo para la generación de controladores difusos basado en una combinación de redes neuronales y algoritmos genéticos. Un modelo de algoritmo genético denominado evolución simbiótica permite la construcción de las reglas difusas y la generación de los conjuntos difusos utilizados como consecuentes en las reglas. Luego un modelo de red neuronal es utilizado para el ajuste de las reglas difusas obtenidas.
El modelo fue validado empíricamente con un problema clásico en el ámbito de sistemas de control: el péndulo invertido.

Palabras clave: Sistemas difusos; Redes neuronales; Evolución simbiótica; Controladores difusos.

* - Este trabajo fue parcialmente financiado por la Universidad de las Naciones Unidas y el International Centre for Theoretical Phisycs (Trieste, Italia)

1 - Introducción:

El uso de controladores difusos surgió a comienzo de los '70, en un intento por diseñar controladores para sistemas que son difíciles de modelar a causa de su no-linealidad y otros problemas[1][2]. Con el transcurso del tiempo, han aparecido distintas aplicaciones de controladores difusos, incluso en ámbitos como finanzas. Pero la aplicación actual mas importante son los procesos de control basados en lógica difusa: control de equipos de aire acondicionado, lavarropas, aspiradoras, control de vehículos (aviones, autos, etc.), etc.

La construcción de controladores difusos es una tarea compleja; requiere definir cuidadosamente las funciones de pertenencia y el conjunto de reglas difusas. En algunos casos relativamente sencillos, el sistema difuso puede ser definido "a mano", ajustando cuidadosamente el conjunto de parámetros del sistema. Pero, si el sistema a ser controlado, o una aproximación de éste, puede ser simulado en una computadora, es conveniente utilizar un algoritmo que determine automáticamente el conjunto de parámetros a controlar y ajustar.

El resurgimiento del interés en las redes neuronales ha inyectado nueva fuerza al campo de los sistemas de control difuso. Aunque las motivaciones e inspiraciones para los sistemas difusos y las redes neuronales son diferentes, existen algunas similitudes que hacen surgir naturalmente su interacción.

Los sistemas difusos están basados en la forma en que los humanos manejamos la información inexacta. Las redes neuronales intentan modelar la arquitectura interna del cerebro. Los sistemas difusos se definen a través de funciones de pertenencia no lineales simples, las cuales asignan grados de pertenencia a los datos de entrada. Los sistemas difusos combinan conjuntos difusos con reglas difusas para producir un comportamiento no lineal complejo. Las redes neuronales basan su potencia en elementos de procesamiento simples que, al ser combinados, tienen, en general, un comportamiento no lineal complejo. Ambos tienen la capacidad de modelar sistemas no lineales complejos con un grado arbitrario de precisión. A causa de estas similitudes, ambos pueden ser combinados para obtener los sistemas neuro-difusos (o redes neuro-difusas), los cuales se aplican, principalmente, en el área de controladores difusos.

Las redes neuro-difusas tienen ventajas que las hacen muy atractivas, tales como la capacidad de aprendizaje y la tolerancia a las fallas. Así, las redes neuro-difusas tienen la capacidad de aprender los conjuntos y las reglas difusas, utilizando algoritmos de aprendizaje diseñados para redes neuronales para el ajuste de los parámetros del sistema.

Diferentes algoritmos para generar controladores difusos han sido propuestos en los últimos años[3][4][5]. Algunos de los mas exitosos están basados en técnicas de computación evolucionaria[12]. Estos algoritmos "evolucionan" estructuras (subconjuntos de parámetros del sistema) hasta encontrar la mas adecuada para resolver el problema presentado.

Una idea interesante consiste en utilizar técnicas de computación evolucionaria para "evolucionar" sistemas difusos representados como redes neuronales[12]. Una nueva aproximación es la llamada "Evolución simbiótica"[6][7], en la cual cada individuo de la población es una unidad de una red neuronal. Las soluciones completas (es decir, las redes neuronales) se obtienen combinando grupos de estas unidades individuales. La bondad de cada individuo se determina por su grado de cooperación con los otros individuos usados para construir las soluciones completas.

En este trabajo, presentamos un modelo de sistema de generación de sistemas difusos basado en GARIC (Generalised Approximate Reasonning-based Intelligent Control)[8], al cual se le agrego la capacidad de generar reglas difusas utilizando el modelo de Evolución Simbiótica. Además, se modificó el cálculo del error propuesto en GARIC, definiendo un conjunto de reglas difusas que permiten medir el error en el sistema.

El sistema fue validado con una aplicación típica de un sistema de control dinámico: el problema del balanceo del péndulo invertido (cart-pole balancing system).

2 - Controladores difusos

Un controlador difuso básico esta formado por tres componentes principales: el fuzificador, la base de reglas y el motor de inferencia y el defuzificador.
El fuzificador convierte datos de entrada desde un espacio de datos predefinido en valores lingüísticos adecuados para los conjuntos difusos utilizando funciones de pertenencia para las entradas predefinidas.
La base de reglas consiste de un conjunto de reglas lógicas difusas del tipo "IF-THEN" para describir la política de control basada en el conocimiento del experto.
El motor de inferencia presenta la salida del fuzificador a la base de reglas para llevar a cabo la implicación lógica y el razonamiento aproximado pata obtener la acción difusa de control.
Por último, el defuzificador obtiene la acción de control a ser aplicada al sistema objeto a partir de la acción difusa inferida por el motor de inferencia utilizando funciones de pertenencia para la salida predefinidas.

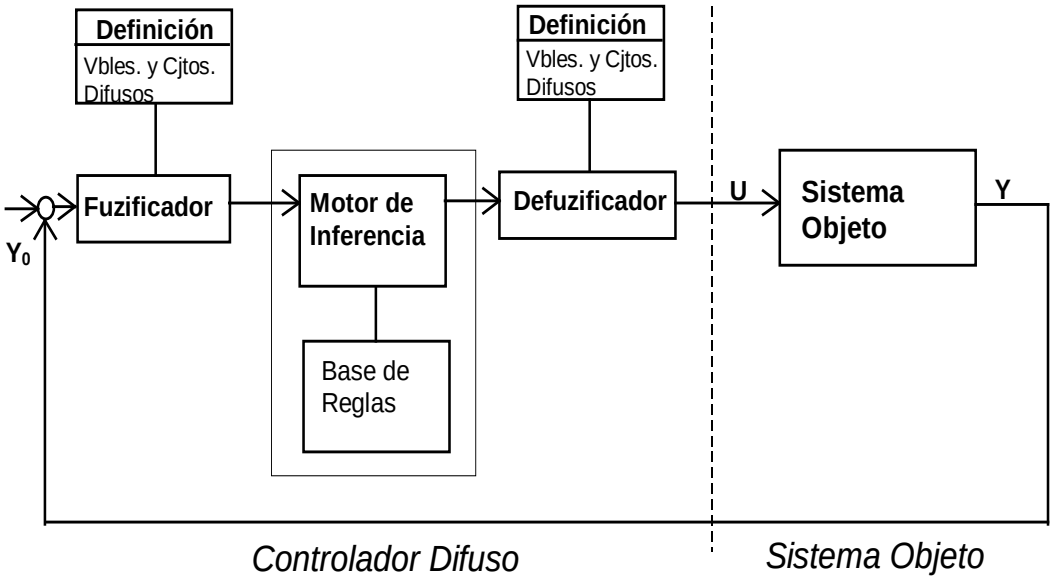

Figura 1: Sistema de Control Difuso

3 - Sistemas neuro-difusos de inferencia

Los sistemas difusos de inferencia están basados en los conceptos de la teoría de conjuntos difusos, reglas difusas y razonamiento difuso[9].
Cuando un sistema difuso tiene como entradas y salidas valores reales, implementa un mapeo no lineal de su espacio de entrada en su espacio de salida. La ventaja de estos sistemas consiste en que son capaces de tratar con valores lingüísticos. Por ejemplo, se puede definir la velocidad de un vehículo como baja, media o alta, dando la definición de las funciones de pertenencia respectivas.
Para definir un sistema difuso es necesario definir las reglas difusas y las funciones de pertenencia para representar los valores lingüísticos. Su selección influencia fuertemente la calidad del sistema.
Las redes neuronales artificiales son estructuras que consisten de elementos simples de procesamiento que se conectan a través de conexiones con pesos asociados[10]. Son capaces también de aproximar funciones o realizar otras tareas aprendiendo de ejemplos. Usualmente el tiempo de aprendizaje es largo, aunque una vez que están

entrenadas su respuesta es casi inmediata.
Una combinación de ambas aproximaciones ofrece la posibilidad de combinar las ventajas de ambos métodos. La red neuronal se puede basar en el conocimiento definido por el sistema difuso, y refinarlo a través de un algoritmo de entrenamiento. No se requiere un largo tiempo de entrenamiento, ya que la red parte con un conocimiento incorporado. Así, la decisión inicial de las funciones de pertenencia y las reglas no se torna crucial, ya que pueden ser ajustadas a medida que se utiliza el sistema.
Hay diferentes modelos de aprendizaje. Algunos de ellos necesitan pares de entrada/salida (en aprendizaje supervisado) y otros trabajan sólo con una señal de refuerzo (un método de aprendizaje no supervisado), que reducen el costo cuando los pares de entrada/salida son difíciles de obtener.

3.1 - El modelo GARIC

En esta sección brevemente describiremos la arquitectura GARIC (Generalised Approximate Reasonning-based Intelligent Control)[8]. Esta compuesto por dos redes neuronales especiales llamadas ASN (Action Selection Network) y AEN (Action Evaluation Network) como se muestra en la *figura 2*.

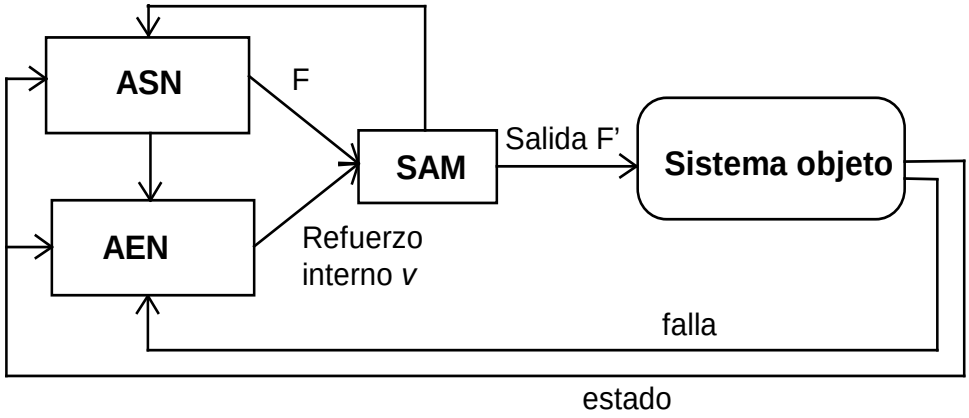

Figura 2: Arquitectura del sistema GARIC

La ASN es una red neuro-difusa que se comporta como un controlador difuso. La AEN estima la bondad del sistema (el refuerzo interno) basado en el estado del sistema y una señal booleana de falla. La salida de ambas redes es combinada por el SAM (Stochastic Action Modifier) para producir la salida real a ser aplicada al sistema objeto. Ambas redes son entrenadas durante el proceso. La AEN actualiza sus pesos para predecir el refuerzo asociado con el estado del sistema. La ASN actualiza sus pesos para que el sistema pueda alcanzar un mejor estado.
Asumimos que hay **n** variables de entrada $\mathbf{X}_i$ ($1 \leq \mathbf{i} \leq \mathbf{n}$) y $\boldsymbol{K}_j$ ($1 \leq \mathbf{j} \leq \mathbf{m}$) valores lingüísticos para cada variable $\mathbf{X}_i$, y $\boldsymbol{R}$ reglas.
La estructura de la ASN mostrada en la *figura 3* tiene cinco capas. La primera capa contiene un nodo por cada variable de entrada $\mathbf{X}_i$. La segunda capa contiene un nodo por cada valor lingüístico μ_{ik} ($1 \leq \mathbf{i} \leq \mathbf{n}$, $1 \leq \mathbf{k} \leq \boldsymbol{K}_i$) de las variables de entrada. La tercera, un nodo que representa a cada regla $\boldsymbol{R}_r$ ($1 \leq \mathbf{r} \leq$ R). La cuarta, un nodo por cada valor lingüístico de la variable de salida. Y la quinta, un nodo por cada variable de salida.
Los nodos de la primera capa solo pasan el valor real que reciben como entrada a los nodos de la segunda capa, donde son fuzificados y estos valores enviados a la tercera. La tercera capa calcula la operación diferenciable **softmin** de sus entradas para obtener $\boldsymbol{w}_r$. Esta operación no corresponde con la definición de T-norm, como usualmente es,

pero tiene la ventaja de ser diferenciable, lo que permite que su utilización en una red de backpropagation. La cuarta capa defuzifica estos valores calculando la operación inversa $\mu^{-1}(w_r)$. La quinta obtiene la salida real haciendo la suma ponderada de sus entradas.

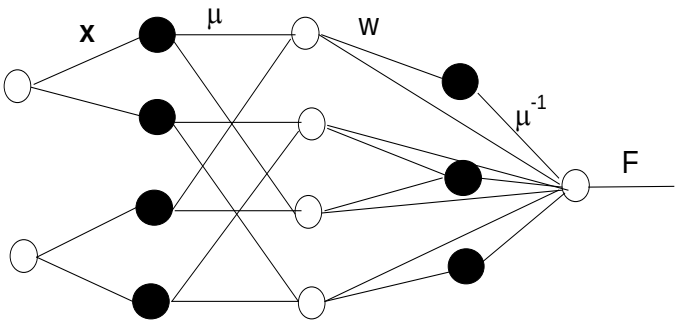

Figura 3: Arquitectura de la ASN

La AEN predice el refuerzo asociado con los diferentes estados del sistema. Es una red de dos capas normal y es entrenada utilizando el algoritmo de backpropagation estándar.
El SAM genera la salida F' a ser aplicada al sistema a partir de la salida F de la ASN y la señal de refuerzo v producida por el AEN. F' se calcula como una variable aleatoria Gausiana con media F y desviación estándar e^{-v}. Con esto se obtiene una mejor exploración del espacio de estados y mejora la habilidad de generalización.
GARIC puede ajustar las funciones de pertenencia de los antecedentes y de los consecuentes, pero no puede generar las reglas.

4 - Computación Evolucionaria

Los algoritmos genéticos [11][12] son una técnica de búsqueda global en la cual las posibles soluciones (llamadas individuos) son codificadas como cadenas (llamadas generalmente cromosomas). Cada solución posible es evaluada sobre el problema específico. Utilizando la operación llamada *crossover* se permite que distintas cadenas sean combinadas, obteniendo así mejores individuos. Existe, además, el operador de *mutación*, utilizado para aplicar una pequeña alteración a las cadenas.
Si una red neuronal puede ser representada como una cadena, entonces un algoritmo genético puede evolucionar poblaciones de redes neuronales, evaluándolas y asignándoles un fitness (medida de su bondad para resolver el problema). Las mejores redes neuronales serán combinadas para obtener nuevas redes neuronales.
Este esquema es muy interesante, porque la mayoría de los algoritmos para redes neuronales sólo buscan el conjunto adecuado de valores para sus parámetros (los pesos), pero con la estructura de la red dada de antemano. Para que estos algoritmos sean exitosos, los valores deben existir en el espacio de búsqueda de la red.
Un problema importante, en el ámbito de los algoritmos genéticos, es la prematura convergencia de la población hacia soluciones subóptimas. A partir de esta convergencia, la búsqueda se hace lineal en el espacio de soluciones haciendo sólo mutaciones, en vez de una búsqueda paralela mas eficiente[6].
Muchas de las alternativas propuestas para resolver este problema requieren, generalmente, del uso de operaciones externas que son computacionalmente costosas, o el uso de técnicas de búsqueda menos eficientes.
Una alternativa muy promisoria es la llamada Evolución Simbiótica, propuesta por Miikkulainen y Moriarty en [6].

4.1 - Evolución Simbiótica. SANE y H-SANE

La idea básica de la evolución simbiótica es que los individuos de la población representan soluciones parciales. El objetivo de cada individuo es ser una solución parcial adecuada para ser combinada con otros individuos de la población. Como cada individuo no es una solución por sí mismo, debe combinarse con otros para obtener un fitness mas alto; es decir, debe mantener una relación simbiótica[6].
Cada solución parcial debe estar especializada en un aspecto del problema. Así, es imposible que ocurra una convergencia prematura, manteniendo diversa la población.
El algoritmo SANE (Symbiotic Adaptive Neuro Evolution) fue propuesto en [6], donde se mostró que es superior a otros métodos tales como el Heuristic Adaptive Critic (un algoritmo por refuerzo) y el GENITOR (un algoritmo genético estándar).
La idea Fundamental de SANE es codificar cada unidad de una red neuronal como una cadena (cromosoma). El fitness de una unidad se determina en base a su grado de cooperación con las otras unidades utilizadas para formar la red. SANE mantiene la población de cadenas que representan las unidades ocultas de una red neuronal feed-forward estándar. Las unidades de entrada y de salida son determinadas por el problema.
SANE es muy efectivo para hacer búsquedas rápidas en el espacio de soluciones, pero se le hace difícil alcanzar la mejor solución[7]. Métodos que evolucionan redes en vez de unidades (como GENITOR), pueden obtener soluciones de mejor calidad.
H-SANE (Hierarchical SANE) es propuesto por los mismos autores en [7] y soluciona esta dificultad. Combina las ventajas de evolucionar redes con las de la evolución de las unidades. H-SANE mantiene dos poblaciones diferentes, una para las unidades y otra para los prototipos de las redes. La población de unidades evoluciona hacia buenas unidades mientras que la población de redes en buenas combinaciones de unidades.
En el Sistema Neuro-Genético presentado en este trabajo, no todas las unidades ocultas son evolucionadas, ya que asumimos que las variables difusas para los antecedentes están definidas previamente. Por este motivo solo se deben codificar las unidades que representan a las reglas y a las variables difusas (con sus respectivos conjuntos difusos) para los consecuentes (capas 3 y 4 de la FAEN).
En esta nueva codificación para H-SANE, cada cadena representa una unidad de la cuarta capa junto con todas las unidades de la tercera capa a las que está conectada y todas las conexiones entre ellas. Por este motivo, es posible representar redes neuro-difusas. Se puede codificar reglas con un número variable de antecedentes y también múltiples reglas pueden utilizar el mismo consecuente.
Esto es una mejora a otras propuestas en las cuales se fijan los antecedentes y los consecuentes y sólo permite que se especifiquen las reglas[4], o que permiten que cada regla tenga sus propios antecedentes y consecuentes[5].
Otra adaptación implementada se basa en el aprendizaje incremental. Una de las principales dificultades que presentan los algoritmos mostrados en [4], [5] y [6] es que sistemas que tienen un comportamiento inestable son aceptados como buenas soluciones.
Para evitar este problema, las soluciones que producen resultados inestables deben ser "castigadas" asignándoles un menor fitness a medida que el tiempo pasa y sus resultados no se acercan al estado estable del sistema[13].

5 - El sistema híbrido neuro-genético para la construcción de controladores difusos

En este trabajo se presenta el Sistema Híbrido Neuro-Genético para la Construcción de Controladores Difusos, que fue implementado en C++, utilizando xforms[14], para ser

ejecutado en sistemas Unix bajo ambiente X11. Es una herramienta de software que permite al usuario construir y evaluar controladores difusos utilizando métodos neuro-genéticos. Esta basada en el modelo GARIC, mejorado con la capacidad de generación de reglas y la posibilidad de utilizar reglas difusas para medir el error del sistema objeto.
Por ejemplo, para la variable difusa VELOCIDAD, sus conjuntos difusos podrían ser: *RAPIDO-ATRAS, ATRAS, DESPACIO-ATRAS, INMOVIL, DESPACIO-ADELANTE, ADELANTE y RAPIDO-ADELANTE*.
Para definir las reglas, el sistema permite especificar los nombres de las variables difusas que están involucradas en los antecedentes (es decir, la condición) y el nombre de la variable difusa que se utilizara como consecuente.
Por ejemplo, una regla podría ser:

> *IF ((VELOCIDAD es RAPIDO-ADELANTE) AND (ANGULO es POSITIVO)) THEN (FUERZA debe ser NEGATIVA-GRANDE)*

Esta regla indica que el sistema se esta moviendo rápidamente en dirección "positiva" y es necesario frenarlo para disminuir la velocidad con la que esta cambiando.
VELOCIDAD, ANGULO y FUERZA son variables difusas definidas en el sistema. ADELANTE, POSITIVO y NEGATIVA-GRANDE son conjuntos difusos definidos para las respectivas variables difusas.
El programa permite definir un sistema difuso completo, lo que incluye conjuntos difusos para cada variable de entrada y de salida y las reglas difusas (*figura 4*).

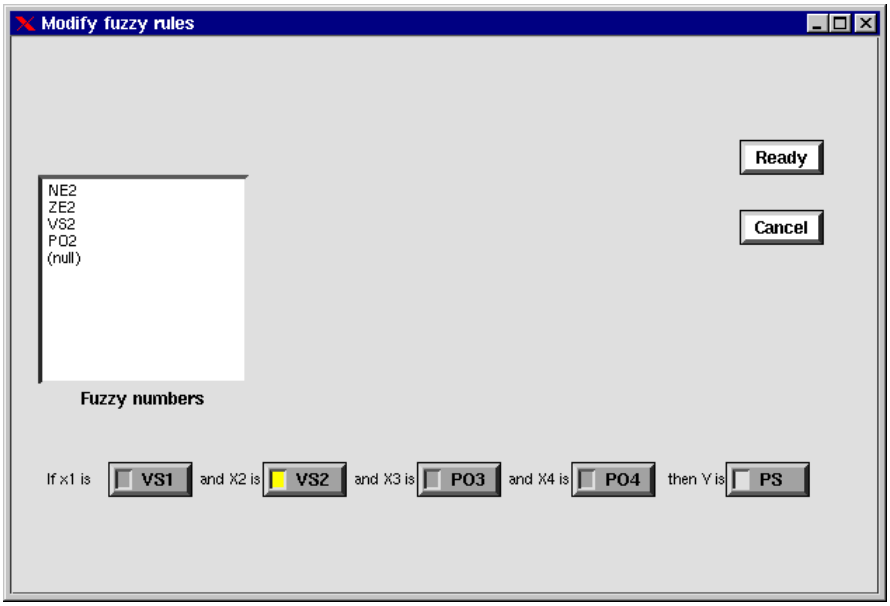

Figura 4: Definición de reglas difusas

Además, permite evaluar números y reglas difusas. Las reglas difusas pueden evaluarse una a una, en forma independiente, o todas juntas (*figuras 5, 6 y 7*).
Una red GARIC se construye a partir de la definición del sistema difuso y entonces se puede ejecutar el algoritmo de aprendizaje para ajustar los parámetros.
La red GARIC se puede construir en forma automática generando las reglas a través del algoritmo de evolución simbiótica propuesto.

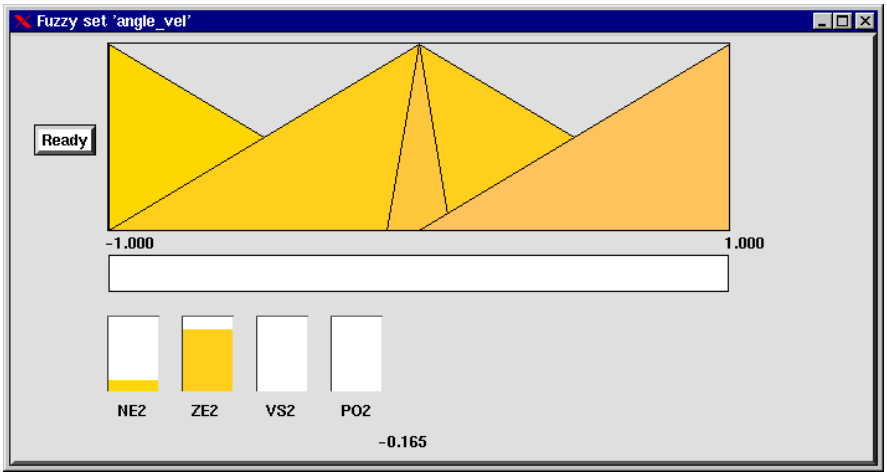

Figura 5: Evaluación de conjuntos difusos

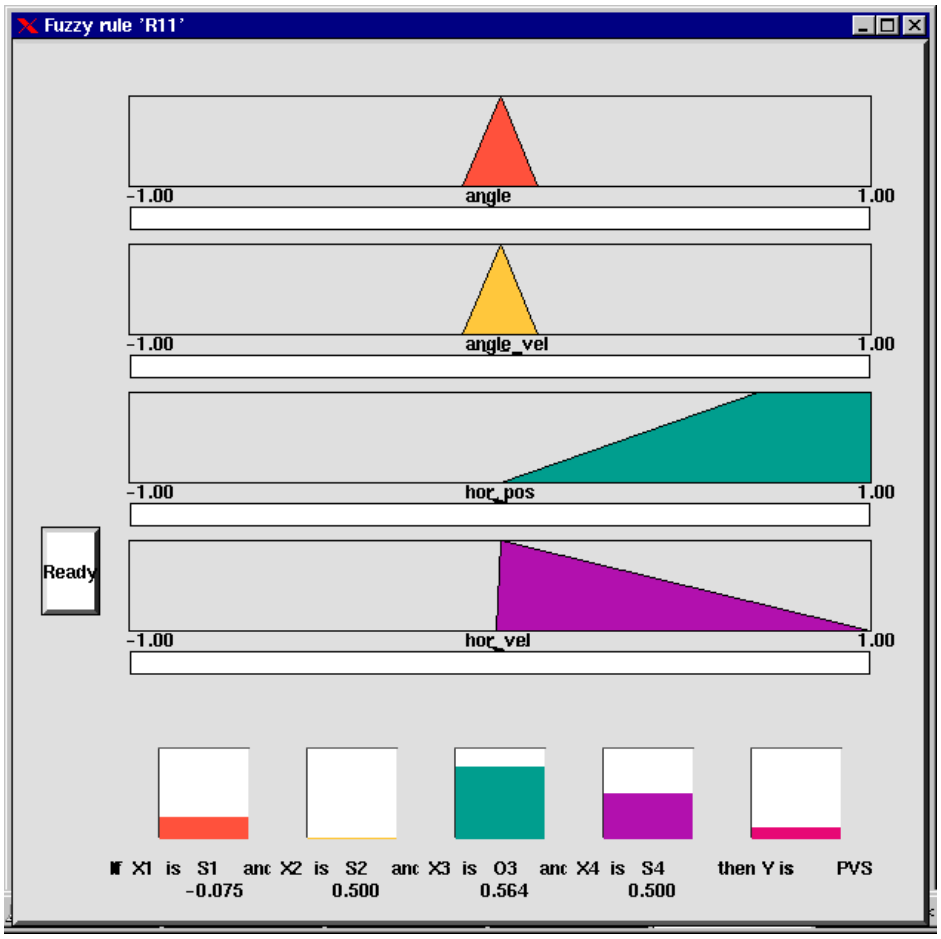

Figura 6: Evaluación de reglas difusas

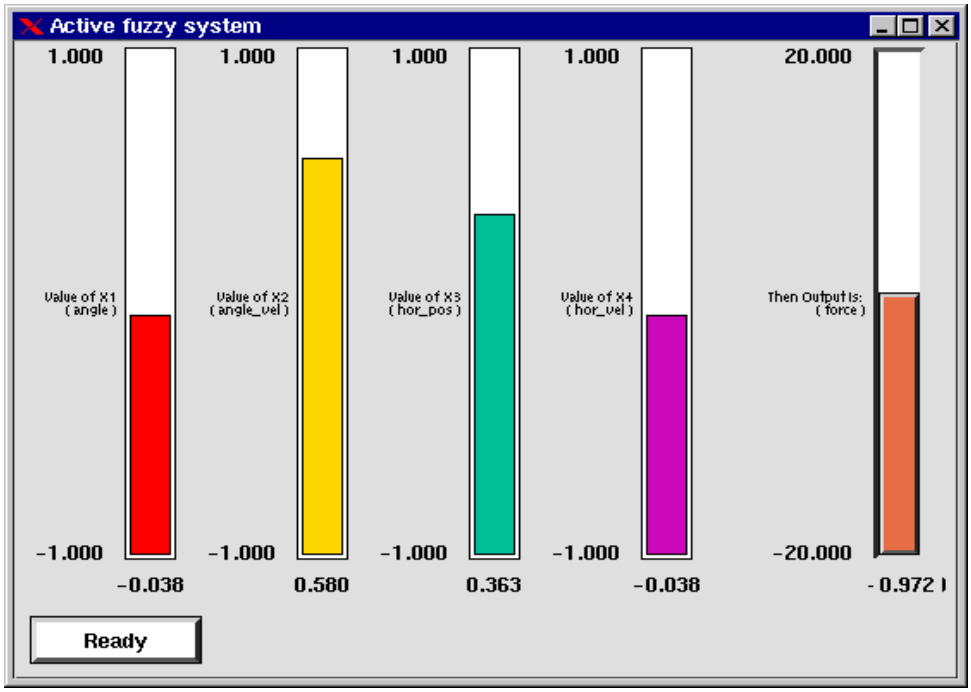

Figura 7: Evaluación de un sistema difuso

5.1 - El modelo GARIC extendido

El modelo extendido definido para su utilización en el sistema es el siguiente:
a) La *FAEN* (Fuzzy Action Evaluation Network) predice una medida de la bondad del estado del sistema, asociado con el estado actual del sistema objeto. La salida es obtenida a través de un proceso de inferencia basado en reglas difusas. Los conjuntos difusos definidos para las entradas y la salida son triangulares. La función básica de la FAEN es la misma que la AEN de GARIC.
b) La EASN (Extended Action Selection Network) basada en la ASN de GARIC se comporta como un controlador difuso. Utilizamos la palabra *Extendido* para indicar que el sistema esta preparado para la generación de reglas difusas. También se modificó el método de aprendizaje, en el original estaba basado en la señal de refuerzo, el cual fue cambiado por un algoritmo de backpropagation utilizando el resultado de la red FAEN como un estimador del error con el que comparar la salida esperada. Otra modificación es que todos los conjuntos difusos son triangulares.
El esquema básico es el mismo que el mostrado en la *Figura 3*. La primera capa recibe las entradas y las distribuye en los nodos de la capa 2.
Los nodos de la capa 2 corresponden a los posibles valores lingüísticos de cada una de las variables de entrada y su función es calcular la pertenencia a través de:

$$\mu(x) = \begin{cases} 1 - |x - c| / der & si\ x \in [c, c + der] \\ 1 - |x - c| / izq & si\ x \in [c - izq, c] \\ 0 & en\ otro\ caso \end{cases}$$

Donde *c*, *der* e *izq* son los parámetros que definen la función triangular de pertenencia.
Cada nodo de la capa 3 corresponde a una regla difusa y su función es calcular la conjunción de antecedentes. En lugar de utilizar la función mínimo usual, utiliza la función ***softmin*** con parámetro *k*[8], que es continua y diferenciable. Obtiene como

resultado w_r que es el grado de aceptación de la regla *r*.

$$w_r = \frac{\sum_i \mu_i e^{-k\mu}}{\sum_i e^{-k\mu}}$$

Cada nodo de la capa 4 corresponde a un valor lingüístico de la variable de salida y su función es calcular el valor real utilizando la variable difusa correspondiente. Este proceso involucra la transformación de valores reales y se basa en el método denominado 'media local de máximos'[8].

$$\mu^{-1} = c + \frac{1}{2}(der - izq)(1 - w_r)$$

El nodo de la capa 5 combina las salidas de todas las reglas en un único valor real.

$$S = \frac{\sum_r w_r \mu^{-1}(w_r)}{\sum_r w_r}$$

Los nodos que contienen parámetros modificables por el algoritmo de entrenamiento son los que corresponden a las capas 2 y 4 del modelo.

Algoritmo de entrenamiento

La red ASN está diseñada para trabajar en situaciones en las que el valor esperado no es conocido. En el algoritmo EASN propuesto, si bien el valor esperado no es conocido en forma inmediata, puede ser estimado por la red FAEN. Entonces, se puede aplicar un algoritmo de backpropagation estándar para ajustar los parámetros.
El error de la red EASN se puede calcular en la forma usual, en base a la diferencia la salida *t* producida por la FAEN y la obtenida *s*:

$$E = (t - s)^2$$

El cambio en el parámetro *p* de la red se puede modificar por el descenso del gradiente con velocidad de aprendizaje η, en base a:

$$\Delta p = \eta \frac{\partial E}{\partial p} = \frac{\partial E}{\partial s} \frac{\partial s}{\partial p}$$

Estas derivadas se pueden calcular como sigue:

$$\frac{\partial E}{\partial s} = -2(t - s)$$
$$\frac{\partial s}{\partial p} = \frac{1}{\sum_i w_i} \sum_{v=CON(Rj)} w_j \frac{\partial v}{\partial p}$$

Donde CON(R_j) corresponde a los valores lingüísticos de los consecuentes de la regla R_j..
A partir de estas ecuaciones es posible obtener los valores particulares correspondientes a cada uno de los parámetros de las funciones de pertenencia de los

valores lingüísticos de la variable de salida.

$$\frac{\partial v}{\partial p} = 1 \qquad \frac{\partial v}{\partial der} = \frac{1}{2}(1 - w_r) \qquad \frac{\partial v}{\partial izq} = -\frac{1}{2}(1 - w_r)$$

En forma análoga se puede continuar con el cálculo para las modificaciones de los parámetros de las funciones de pertenencia para los antecedentes, desarrollo que no se presenta debido a su extensión.
El sistema permite habilitar o deshabilitar la actualización de las funciones difusas para los antecedentes (nodos de la capa 2).
En general, no es absolutamente necesario que los nodos de la capa 2 (antecedentes) actualicen sus parámetros. Los nodos de la capa 4 (consecuentes) pueden "aprender" a compensar ciertos errores introducidos en la definición de los antecedentes, tales como conjuntos difusos desfasados o un escaso número de estos conjuntos.

5 - Evaluación del sistema

Los sistemas a ser controlados son implementados independientemente del sistema neuro-genético. Se comunican con el sistema principal a través de pipes. Esto permite simular diferentes sistemas de control sin tener que reconstruir el sistema principal cada vez que esto ocurre.
Esta estrategia libera al usuario de modificar el sistema principal y le permite crear sus propias aplicaciones (con interfaces de usuario de acuerdo a sus necesidades), que simplemente interactúan con el sistema neuro-genético a través del canal de comunicación antes mencionado.

5.1 - El péndulo invertido

El problema del péndulo invertido es el benchmark estándar para este tipo de algoritmos. Consiste en determinar la fuerza a ser aplicada para empujar hacia atrás o adelante un vehículo con un péndulo en posición vertical sobre él. El péndulo debe ser mantenido en su vertical (o cerca de ella) y el vehículo no debe sobrepasar ciertos límites.
Las ecuaciones que definen el modelo son:

$$\ddot{\theta} = \frac{mg.sen\theta_t - \cos\theta_t[F_t + m_p l\,\dot{\theta}_t^{\,2}\, sen\theta_t]}{(4/3)ml - m_p l.\cos^2\theta_t},$$

$$\ddot{\rho}_t = \frac{F_t + m_p l[\dot{\theta}_t^{\,2}\, sen\theta - \ddot{\theta}_t \cos\theta_t]}{m}$$

donde:

$\rho =$ Posición del vehículo

$\dot{\rho} =$ Velocidad del vehículo

$\theta =$ Angulo del péndulo

$\dot{\theta} =$ Velocidad angular del péndulo

l = Longitud del péndulo

m_p = Masa del péndulo

m = Masa del péndulo mas la del vehículo

F = Magnitud de la fuerza

g = Aceleración debida a la gravedad = 9,8

Para simular el sistema se usa el método de aproximación numérica de Euler, utilizando ecuaciones discretas en el tiempo de la forma $\theta(t+1) = \theta(t)+\tau\theta(t)$, con el intervalo de tiempo τ por lo general 0,02 segundos. Para medir la performance se utilizaron los mismos parámetros físicos usados en [3][6][15].
El modelo fue implementado como un sistema ejecutable independiente que se comunica con el sistema principal a través de pipes. La interface es como la muestra la *figura 8*.
En la parte superior izquierda se presentan figuras que representan la posición del vehículo y la del péndulo. Además, hay dos campos que muestran el valor numérico de la posición y el ángulo.
En la parte derecha hay campos para modificar los parámetros físicos del sistema.
En la parte inferior se encuentra un gráfico estadístico que cambia a medida que avanza la simulación e indica la posición del vehículo y el ángulo del péndulo.
Por último, en la parte central, hay una serie de botones para el control del sistema.

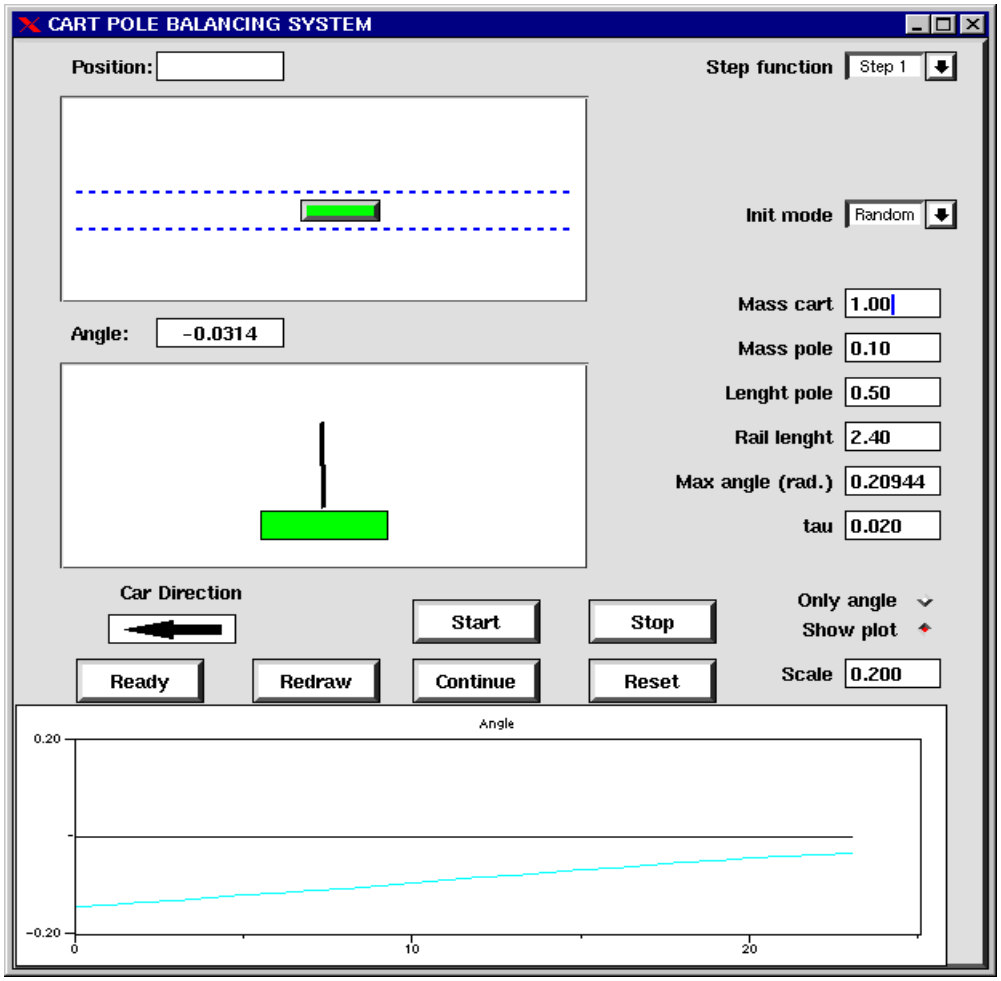

Figura 8: Interface del sistema del péndulo invertido.

5.2 - Resultados

En esta sección presentamos una serie de resultados obtenidos utilizando el proceso de evolución simbiótica descripto antes.
En un primer conjunto de casos de prueba, se consideraron sistemas que balancearon el péndulo sin tener en cuenta la posición del vehículo. Así, los sistemas consisten de dos entradas (ángulo y velocidad angular) y una salida (la fuerza a aplicar).
La posición inicial del péndulo se elige aleatoriamente en el rango [-0.2, 0.2] radianes, y una velocidad angular en el rango [-1.5, 1.5] radianes por segundo.
Cada sistema fue evaluado en 10 corridas de 500 unidades de tiempo. Se utilizó la estrategia de aprendizaje incremental.
La *figura 9* muestra el fitness promedio de las soluciones obtenidas. Se puede ver que buenas soluciones son obtenidas rápidamente. Indicando que el sistema evolucionario puede encontrar muy buenas soluciones en poco tiempo.

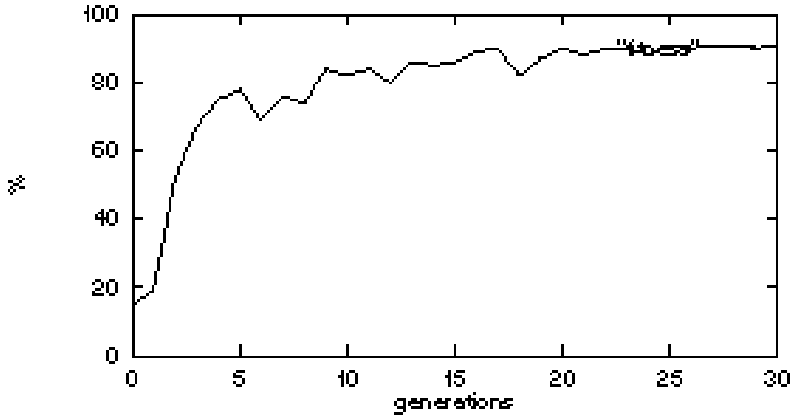

Figura 9: Fitness promedio de las soluciones obtenidas

La *figura 10* presenta los resultados obtenidos al inicializar el sistema en diferentes posiciones elegidas al aleatoriamente. Esta figura muestra que el péndulo puede ser balanceado sin importar el punto de inicio del sistema.

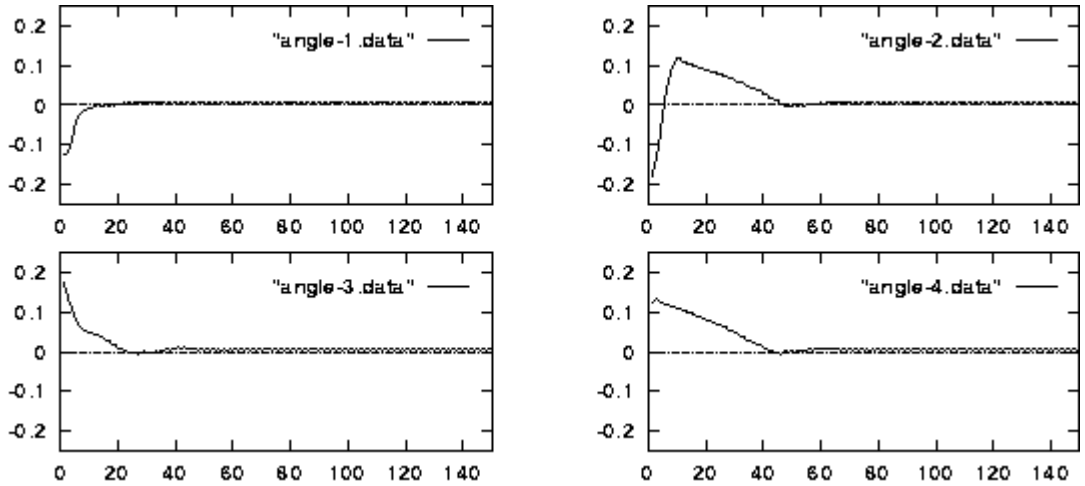

Figura 10: Algunos resultados obtenidos al iniciar la simulación en diferentes posiciones (sistema con dos entradas)

En un segundo lote de experimentos, se consideró el problema de balancear el péndulo y, además, centrar el vehículo. Los sistemas evolucionados utilizando la estrategia

incremental estaban formados por redes de cuatro entradas (ángulo, velocidad angular, posición y velocidad del vehículo) y una salida (la fuerza). El péndulo se inicializó en posiciones aleatorias en el rango [-0.2, 0.2] radianes y la velocidad del vehículo en el rango [-2, 2] metros. Cada sistema fue validado haciendo diez corridas de 3000 unidades de tiempo.
La *figura 11* muestra el valor de fitness contra el número de generaciones en una corrida típica. El proceso del aprendizaje incremental se puede ver en este gráfico. Entre las iteraciones 4 y 12 el fitness no crece demasiado; lo mismo ocurre entre las generaciones 15 y 20; y también desde la generación 25 en adelante.
En la *figura 12* mostramos resultados de diferentes corridas del sistema evolucionado comenzando en posiciones aleatorias. En todos los casos el sistema alcanza soluciones no estables.

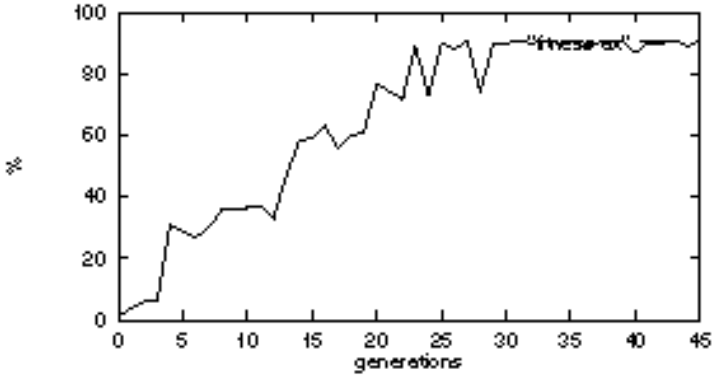

Figura 11: Fitness contra número de generaciones

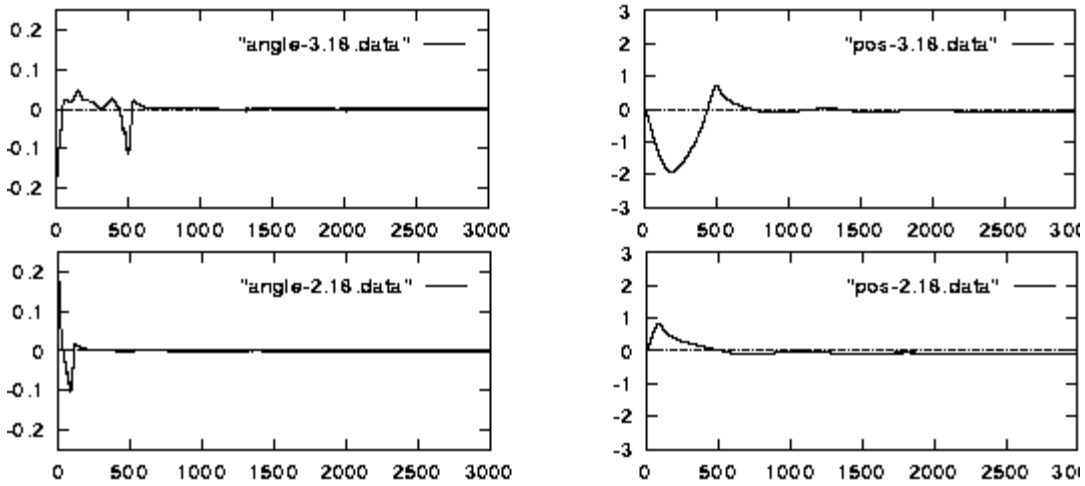

Figura 12: Distintas casos con inicializaciones aleatorias (sistema con cuatro entradas)

6 - Conclusiones

En este trabajo presentamos un nuevo sistema que permite generar y ajustar controladores difusos utilizando una combinación de redes neuro-difusas y evolución simbiótica.
Los resultados obtenidos en el ejemplo descripto en el punto anterior nos permiten apreciar ciertas ventajas. La obtención de sistemas adecuados se logra en menos

tiempo que en otros métodos alternativos. Las soluciones obtenidas se caracterizan además por ser estables. En los experimentos, el tiempo durante el cual fue evaluado el sistema que esta siendo evolucionado no sobrepasó las 3000 unidades de tiempo (considerablemente menor que los 50000 iteraciones utilizados en [6], [8] y [15]).
Esto muestra que la estrategia combinada de evolución simbiótica y redes neuronales es efectiva para construir sistemas difusos para ser aplicados en control. La estrategia incremental propuesta permite obtener soluciones de mayor calidad. Particularmente, en el caso del péndulo invertido, esta combinación provee mejores soluciones que las citadas en la bibliografía.
Otros problemas con los que puede ser validado el sistema incluyen manipulación de robots, problema de estacionamiento, etc.

7 - Agradecimientos

El trabajo fue desarrollado en el contexto de un proyecto de investigación conjunto del Centro Internacional de Física Teórica (ICTP), Trieste, Italia y la Universidad de las Naciones Unidas (UNU), en colaboración con la Universidad Nacional de San Luis (UNSL) y la Universidad de Ciencia y Tecnología (UST), Hefei, China.

Los autores agradecen al Prof. Rinus Verkerk, del ICTP, supervisor del proyecto de investigación en el que este trabajo fue realizado. Además al Prof. Raúl Gallard de la Universidad Nacional de San Luis, director del proyecto investigación en el cual los autores argentinos trabajan y han desarrollado parte del material aquí presentado.

Se agradece también al Prof. Alberto Colavita, Director del Laboratorio de Microprocesadores del ICTP, lugar en el que parte de las actividades aquí presentadas fueron realizadas, así como a todas las instituciones involucradas.

Bibliografía

[1] - P. J. King, E. H. Mamdani. The Application of Fuzzy Control Systems to Industrial Process. Automatics, 13, ps. 235-242.
[2] - E. H. Mamdani, N. Baaklini. Prescriptive Method for Deriving Control Policies in a Fuzzy Logic Controller. Electronics Letters, 11, 625-626.
[3] - D. D. Leitch (1995). A New Genetic Algorithm for the Evolution of Fuzzy Systems. PhD Thesis. Department of Engineer Science, University of Oxford.
[4] - C. L. Karr (1991). Design of a Cart-Pole balancing Fuzzy Logic Controller using a Genetic Algorithm. SPIE Conf. on Applications of Artificial Intelligence, WA.
[5] - M. A. Lee and H. Takagi (1993). Neural Networks and Genetic Algorithms Approaches to Auto-Design a Fuzzy System. FLAI'93.
[6] - D. E. Moriarty and R. Miikkulainen (1996). Efficient Reinforcement Learning through Symbiotic Evolution. Machine Learning, 22:11-32.
[7] - D. E. Moriarty and R. Miikkulainen (1996). Hierarchical Evolution of Neural Networks. Technical Report AI96-242. Department of Computer Sciences, The University of Texas at Austin.
[8] - H. R. Berenji and P. Khedkar (1992). Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. IEEE Transactions on Neural Networks, vol. 3, no. 5.
[9] - Jyh-Shing, Roger Jang and Chuen-Tsai Sun. Neuro Fuzzy Modelling and Control. Proceeding of the IEEE, March 1995.
[10] - L. Fausset - Fundamental of Neural Networks, Architectures, Algorithms and

Control. Prentice Hall . Englewood Clifs, 1994.
[11] - D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA, Addison Wesley.
[12] - Z. Michalewicz (1994). Genetic Algorithms + Data Structures = Evolution Programs 2nd. Edition. Springer Verlang.
[13] - M. Cena, C. Kavka, G. F. Wu and Z. Q. Fu. Fuzzy Systems Generation through Simbiotic Evolution. Enviado a EIS'98, Tenerife, España.
[14] - T.C.Zhao and Mark Overmars. Forms Library, A Graphical User Interface Toolkit for X. 1996, 1997.
[15] - A. G. Barto, R. S. Sutton and C. W. Anderson(1983). Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13:834-846.